

Routing between Visible Vertices in Constrained Theta Graphs

Jonathan Rodríguez

An Honors Thesis for the Department of Computer Science

TUFTS UNIVERSITY

May 2021

ADVISOR: Prof. Diane Souvaine

COADVISOR: Prof. Prosenjit Bose

COADVISOR: Dr. Matias Korman

Acknowledgments

This work is a joint project with Dr. Hugo Akitaya, Andrew Gonczi, my thesis advisor Prof. Diane Souvaine, and my co-advisors Prof. Prosenjit Bose and Dr. Matias Korman. I would like to thank each of them for their insight and guidance. I would also like to thank the rest of the Tufts Computational Geometry Research Group for helpful discussions.

Thanks also to the Tufts Summer Scholars Program and T-Tripods Institute (NSF HDR grant 1934553) for partial support of this work

JONATHAN RODRÍGUEZ

TUFTS UNIVERSITY

May 2021

Routing between Visible Vertices in Constrained Theta Graphs

Jonathan Rodríguez

ADVISOR: Prof. Diane Souvaine

Routing is the process of sending a message from a source vertex to a destination vertex in a network. When we have complete knowledge of the graph in advance, there are many algorithms that find the optimal path (e.g. Dijkstra, Bellman-Ford). It is not always viable, however, to know the whole network. For example, the network could be incredibly large, or the devices could be moving. Constantly updating the whole network can be very expensive, so we want to avoid this expense by using a limited amount of information at each node, namely just the neighbors. Introducing obstacles on the graph further increases the complexity of the problem.

A good way to approach this problem is by using a family of graphs known as constrained Θ -graphs. We propose Spiral Scan Routing, the first competitive routing algorithm on Θ_{4k+2} graphs. This algorithm depends on the source and vertex being visible to each other, but it is deterministic and has a constant stretch factor of $\frac{1}{1-2\sin(\frac{\pi}{4k+2})}$. Spiral Scan Routing, however, does require a message of size $O(n)$ to be passed. We conclude by discussing ideas for improving this algorithm in order to reduce this space requirement.

Contents

Acknowledgments	ii
Abstract	iii
Chapter 1 Introduction	1
1.1 Geometry Background	1
1.2 Graphs	2
1.3 Routing Algorithms	3
1.4 Θ -Graphs	4
Chapter 2 Previous Work	8
2.1 What We Know about Constrained Θ_{4k+2} Graphs	9
Chapter 3 Routing on the Θ_{4k+2} Graph with $O(n)$ Memory	10
3.1 Algorithm Description	14
3.2 Algorithm Analysis	23
Chapter 4 Future Work	27
Bibliography	30

Chapter 1

Introduction

This research is concerned with a fundamental question in computer science: how to efficiently find a short path between two nodes in a graph. We discuss routing, the process of sending a message from a source vertex to a destination vertex, in a setting where each node knows only of its neighbors. The goal is to route using a limited amount of information while still doing it deterministically and efficiently. Specifically, we route on a special family of graphs known as constrained Θ -graphs that can be used to represent real-world networks with obstacles.

In this thesis, we present a deterministic, 1-local competitive routing algorithm between any two visible points in constrained Θ_{4k+2} graphs. We also prove that any path taken by our algorithm spans a length of at most $\frac{1}{1-2\sin(\frac{\pi}{4k+2})}$ times the Euclidean distance between the start and end nodes, which means the spanning ratio is constant.

1.1 Geometry Background

We begin by discussing one of the most fundamental concepts in geometry: lines. For consistency, we shall denote the infinite line between p and q as pq , the line segment with endpoints p and q as \overline{pq} , and a directed segment from p to q as \overrightarrow{pq} . We also describe a ray with endpoint p and another point q as \overrightarrow{pq} , in which case we explicitly refer to it as a ray to avoid ambiguity. A *polygonal chain* is defined as a sequence of points p_1, \dots, p_n and the segments between each pair of consecutive points in the sequence. A polygonal chain is *simple* if it has no self-intersections.

We assume points are in *general position*:

1. no three points are collinear, and
2. no two points share an x -coordinate or a y -coordinate.

These assumptions prevent degenerate cases that can be easily circumvented. For instance, if the second assumption does not hold, the plane can be rotated slightly so that it does. Moreover, a random point set, such as a set of points uniformly distributed in the unit square, is in general position with probability 1.

1.2 Graphs

A graph $G = (V, E)$ is defined as a pair of sets V and E , where V is the set of vertices and E is the set of edges. An edge is a two-element subset of V . A weighted graph is defined as $G = (V, E, w)$, where the parameter w is a function that maps edges to real numbers. In applications, the weight function can represent time, cost, or, for the purposes of this research, distance.

A graph can be *directed* or *undirected*. If it is directed, then each edge is an ordered pair, namely $E \subset V \times V$. That is, (v_1, v_2) and (v_2, v_1) are two different edges. In the undirected case, $\{v_1, v_2\} = \{v_2, v_1\}$.

Two vertices sharing an edge are *adjacent*. Given a vertex v , the set of all vertices adjacent to v is the *neighborhood* of v . A *walk* is a sequence of adjacent vertices. In this thesis, we use *path* to mean the same as walk.

In the unweighted setting, the path from u to v that uses the fewest number of edges is the shortest path. In a weighted graph, the shortest path is the path whose sum of edge weights is minimized. Algorithms such as Dijkstra's algorithm or the Bellman-Ford algorithm are common ways to find the shortest path from u to v .

A *geometric graph* is one where the vertices are points on the plane and each edge is a straight segment between two vertices. A weighted geometric graph is called *Euclidean* when, for each edge $e = \{u, v\}$, $w(e)$ is the Euclidean distance between u and v .

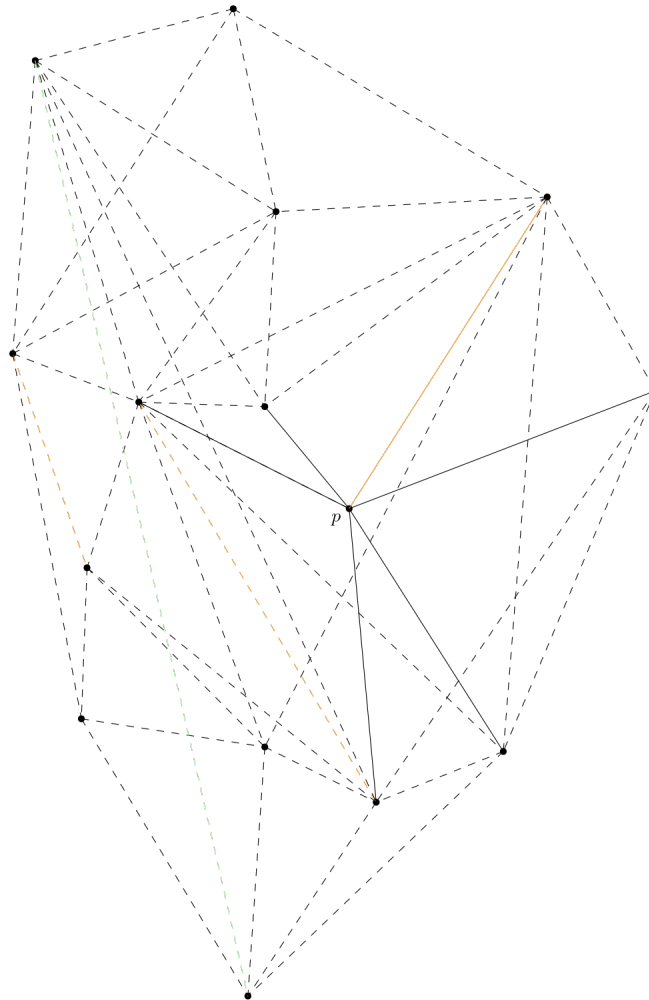


Figure 1.1: If we were to traverse this graph using a local algorithm, then the only information available at site p would be the solid edges and the sites at the ends. We would have no knowledge of the dashed edges or other sites in the graph.

1.3 Routing Algorithms

Routing is the process of sending a message from a source vertex to a destination vertex in a graph. The source s is the vertex where the algorithm begins, and the destination or target t is the goal. The message is sent from one vertex to another following edges of the graph. To limit ambiguity, we will refer to the vertices of the graph being traversed as “sites” rather than “vertices” or “points”.

A routing algorithm is called *local* if the only information available to the algorithm at each site is its neighborhood [Figure 1.1]. A routing strategy that uses just local information is space efficient.

Local algorithms minimize the space used, but if the graph is dense, the neighborhood of

each site can be $\Omega(|V|)$. In order to further reduce the space constraint, subgraphs that approximate the dense graph are used. More formally, for any graph G , a subgraph H is a t -spanner if, for any two points in G , the shortest path between them in H is at most t times the length of the shortest path between them in G , for some $t \geq 1$. In this context, the smallest such t is the *spanning ratio* or *stretch factor*. Some examples of spanners of the complete Euclidean graph include Delaunay triangulations (with stretch factor less than 1.998 [Xia13]) and Θ -graphs (which we will discuss in the next section).

A routing algorithm is *competitive* if we can guarantee that the distance traversed from s to t is at most a constant factor of the shortest path from s to t on the graph. Specifically, an algorithm is c -competitive if the path traversed by the algorithm is at most c times the length of the shortest path.

1.4 Θ -Graphs

Before diving into Θ -graphs, it is worth discussing the *visibility graph*. Given a set of points V , we define a set of constraints T , where every element of T is a segment between two points of V . Given two points u and v , u is *visible* to v if and only if $\overline{uv} \in T$ or \overline{uv} does not intersect the interior of any segment of T . The visibility graph of V with respect to T is the graph $G = (V, E)$ where E is the set of edges e with visible endpoints in V .

Θ -graphs were introduced independently by Clarkson [Cla87] and Keil & Gutwin [KG92]. Given a set of points V , the Θ_m graph is defined as follows. For each vertex $v \in V$, split the plane radially into m cones of equal angle [Figure 1.2] and add the edge to the vertex closest to v in each cone. Here, “closest” refers to the vertex in the cone whose projection onto the cone’s bisector is the smallest distance from v . All of the cones have angle $\frac{2\pi}{m}$ in a Θ_m graph. When m is clear from the context, we use α to denote this angle. Note also that varying m can lead to different graphs on the same point set [Figure 1.3].

Given a site p of the Θ_m graph, in each cone there may be several edges adjacent to p . Consider an edge between p and q . If q is the site closest to p in its cone (following the definition of “closest” discussed earlier), then we call this an *outgoing edge* from p . If p is the closest to q in its cone, we call this an *incoming edge* to p . Of course, if both of these conditions hold, then the

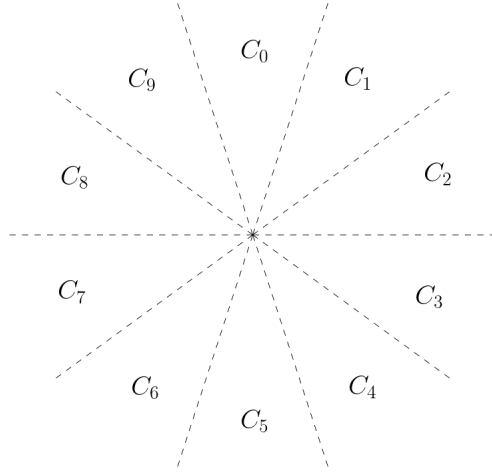


Figure 1.2: Division of cones in a Θ_{10} graph at a single vertex. By convention, we will always orient them so that the bisector of C_0 is vertical.

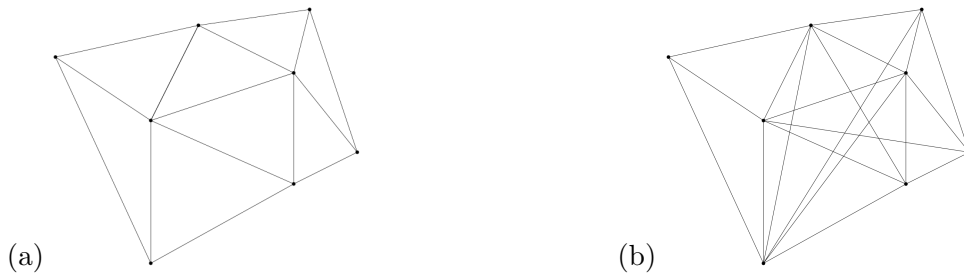


Figure 1.3: Two Θ graphs on the same set of vertices. (a) is the Θ_6 graph and (b) is the Θ_{10} graph.

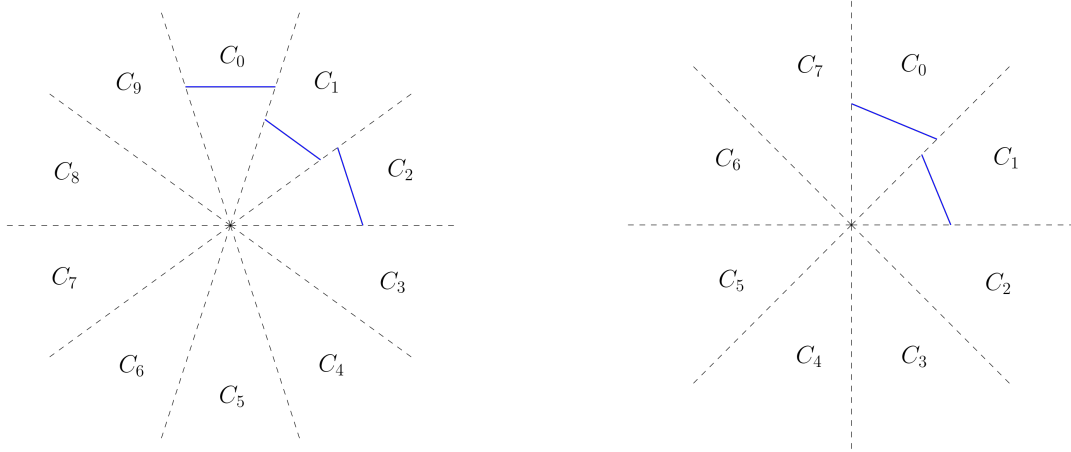


Figure 1.4: A comparison between the sweeplines in Θ_{10} and Θ_8 graphs. Note that the sweeplines in the Θ_{10} are parallel to cone boundaries. This property is shared by all Θ_{4k+2} graphs, Other Θ -graphs, such as Θ_8 graphs, do not have this property.

edge is both outgoing and incoming. This can be thought of as a single bidirectional edge or as two overlapping edges. It is also worth noting that if \vec{pq} is an outgoing edge from p , then it is also an incoming edge to q , and vice versa. The distinction between incoming edges and outgoing edges can give direction to the Θ graph.

The two rays that define a cone are called its *boundaries*. In a Θ_m graph, we denote the i^{th} cone ($0 \leq i < m$) of site p as C_i^p . If p is clear from context, we will only use C_i . In this thesis, we follow the convention that C_0 is directed upward with a vertical bisector, and i increases in clockwise order around the apex. Note also that the orientation of the cones is consistent from all points. More formally, if $\vec{vp_1}$ and $\vec{vp_2}$ are the boundaries of C_i^v , and $\vec{v'p'_1}$ and $\vec{v'p'_2}$ are the boundaries of $C_i^{v'}$, then $\vec{vp_1}$ is parallel to $\vec{v'p'_1}$, $\vec{vp_2}$ is parallel to $\vec{v'p'_2}$, and $\angle p_2vp_1 = \angle p'_2v'p'_1$.

Given a site p and a site $q \in C_i^p$, the *sweepline* through q is the line segment orthogonal to the angle bisector of C_i^p , with endpoints lying on the cone's boundaries. The *canonical triangle* from p to q is the triangle defined by the boundaries of C_i^p and the sweepline through q . We use Δ_{pq} to denote this.

Θ_{4k+2} graphs have the key property that the sweepline of any cone is parallel to another cone's boundary [Figure 1.4], for $k \geq 1$. If the angle of each cone is α , then the angle made by the sweepline with either of the boundaries of the cone is $\frac{\pi-\alpha}{2}$. Since $\alpha = \frac{2\pi}{4k+2}$, this angle is equivalent to $\frac{2k\pi}{4k+2}$, which is a multiple of α

A *constrained Θ_m graph* is defined in a way similar to the Θ_m graph, except we start with

a set of vertices and a set of constraints. A constraint in this context is the same as that of the visibility graph: a segment between two points of the vertex set. The constrained Θ_m graph is constructed by adding an edge to the nearest visible vertex in each cone. At each endpoint of a constraint, the constraint splits the cone into subcones [Figure 1.5]. An edge is added to the closest visible vertex in each subcone (this might be the other endpoint of the constraint). It is worth noting that even when looking in subcones, the closest vertex is determined by the projection onto the angle bisector of the full cone, not the subcone.

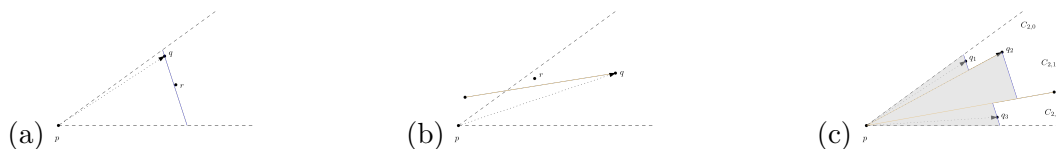


Figure 1.5: Three examples of special cases when tracing an edge from p . In (a) note that the Euclidean distance from p to r is less than that from p to q , but following the definition of the Θ -graph, the edge goes to q . In (b), even though r is closer to p , it is not visible to p so it is ignored. In (c), since p is the endpoint of constraints it has multiple outgoing edges in the cone, one per subcone. An important subtlety is that in each subcone, the sweepline is orthogonal to the bisector of the full cone, not the subcones.

We say that a segment intersects a cone if there is a point on the interior of the segment that is strictly between the cone boundaries. Note that if a segment lies on a cone boundary, then it does not intersect any cone. We also extend this terminology to polygonal lines.

When dealing with constrained Θ -graphs, we add a few assumptions to the standard definition of general position for simplicity. These are as follows:

1. no site lies on the boundary of another site's cone, and
2. no two constraints intersect.

The first assumption can be achieved by rotating the plane ever so slightly. The second assumption is to remove a set of constructions that would make routing impossible. If constraints can intersect, we could easily construct a case where the target t is surrounded by constraints and no site is visible to it, so we avoid these cases.

Chapter 2

Previous Work

Greedy routing tends to work well in Θ -graphs. The routing algorithm is simple: at each step, take the outgoing edge in the cone that contains the target t . This algorithm routes competitively to on all Θ_m graphs with $m \geq 7$ [RS91]. For such Θ -graphs, each step guarantees that the distance to t decreases, so eventually we will reach the destination. This argument does not work for Θ_5 , for example, since it is possible that the distance to t increases even when we go in the cone that contains it [STMZ18]. There are also known competitive algorithms for routing on Θ_4 graphs [BDCHS19] and Θ_6 graphs [BFVRV12].

Greedy routing may fail with constraints. It is simple to construct an example such as in Figure 2.1, where the cone containing t has no edges at all. Thus, we are in need of a method of circumventing constraints.

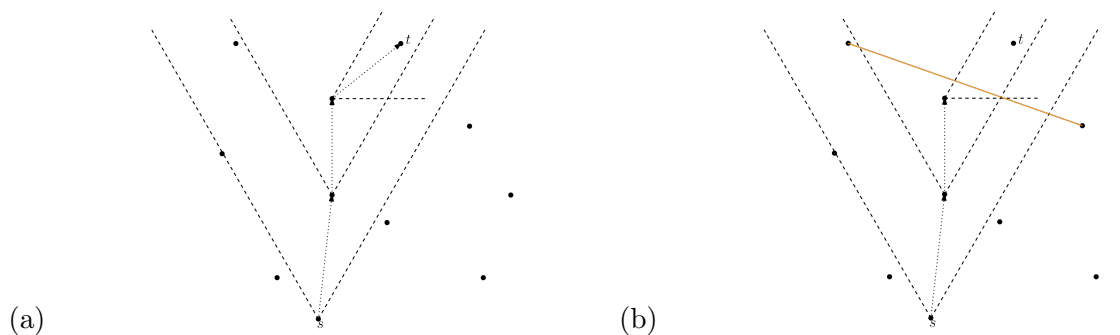


Figure 2.1: (a) shows greedy routing in Θ_6 successfully reaching t . When we introduce constraints in (b), we lose the guarantee of reaching t , since we could have a constraint blocking off the cone that contains t .

2.1 What We Know about Constrained Θ_{4k+2} Graphs

Bose et al. [BFVRV15] provide an algorithm for routing in the constrained Θ_6 graph. Besides that, Bose & Van Renssen [BVR14] have shown that the constrained Θ_{4k+2} is a $1 + 2 \cdot \sin(\frac{\alpha}{2})$ -spanner of the visibility graph, where α is the angle of the cone. That is, given two points on the visibility graph, the shortest path between them on the constrained Θ_{4k+2} graph is at most $1 + 2 \cdot \sin(\frac{\alpha}{2})$ times the length of the shortest path on the visibility graph itself. Note that this is a constant spanning ratio, since α depends only on the number of cones. This is ideal as the spanning ratio does not scale with the number of vertices in the graph. Furthermore, when $k > 1$, the spanning ratio decreases as the number of cones increase, so as k increases, the constrained Θ_{4k+2} graph becomes a better approximation of the visibility graph. However, to our knowledge no deterministic algorithm has been developed for routing on constrained Θ_{4k+2} graphs for $k > 1$.

Chapter 3

Routing on the Θ_{4k+2} Graph with $O(n)$ Memory

We now discuss our strategy for routing between two visible vertices s and t in the constrained Θ_{4k+2} graph. We will then prove that this strategy is competitive, by proving a bound of $\frac{1}{1-2\sin(\frac{\pi}{4k+2})} \cdot |st|$ for the length of the path taken. For convenience, we assume at each step that our current site p is to the right of the directed segment \overrightarrow{ts} (unless we are at s). If we ever cross st , we can flip the coordinate plane to maintain this assumption.

We start by discussing the kind of situation we try to avoid in our algorithm. Suppose we are routing to t and we are currently at site p . Given that we only know about the neighbors of p , how do we pick which edge to take? Ideally, we would want to take one that brings us closer to t , as we would do for greedy routing. This approach, however, would not always work, as discussed previously with the constrained Θ_6 graph in Figure 2.1. If the cone that contains t is completely blocked by a constraint, we are faced with the difficult decision of guessing which endpoint of the constraint is closer. If we guess wrong, we may have to travel along a path of unbounded length. We seek to avoid this decision by introducing the witness line.

The *witness line* is a simple polygonal chain from t to our current location used to guide our routing strategy. We denote the initial witness line as W_0 , and the witness line after j steps as W_j . The witness line is maintained in a way that prevents it from being crossed by constraints, so intuitively, if the algorithm were allowed to “walk” freely on the plane, it could follow the witness

line to reach t . However, the witness line does not use edges of the graph, so instead we approximate this idea by “hugging” the witness line, circling around any constraints that block the current site from t . The witness line satisfies the following properties:

1. W_j is simple (i.e., it has no self crossings).
2. W_j is a polygonal chain whose first edge is contained in \overline{st} . Moreover, all other edges are along some cone boundary.
3. The witness line is a convex chain. That is, three consecutive vertices along W_j form a clockwise turn [Figure 3.1(a)].
4. No constraint can properly cross W_j [Figure 3.1(b)].

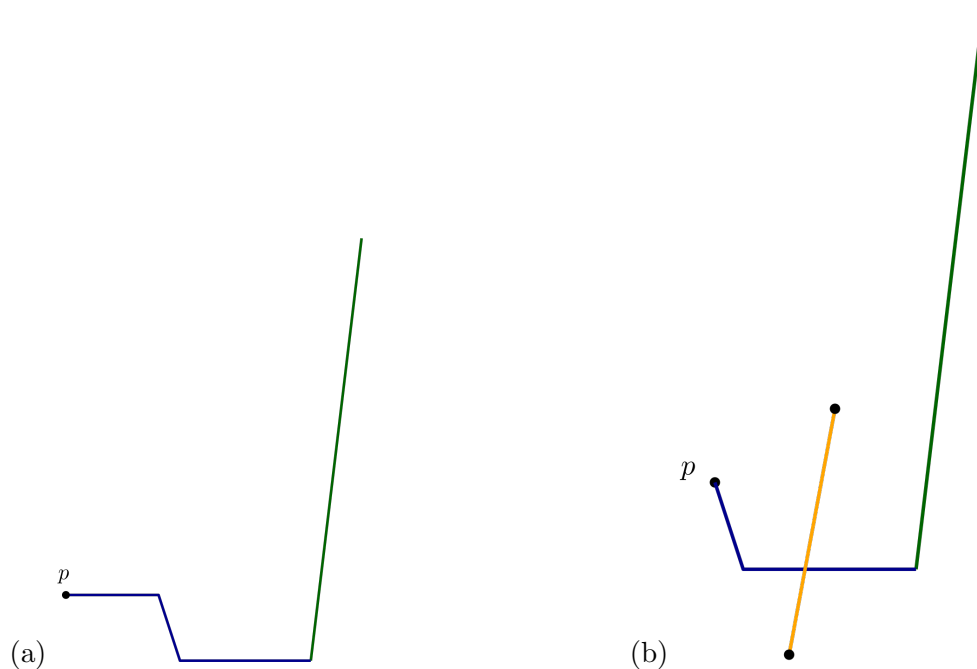


Figure 3.1: Examples that the last two invariants guarantee will never occur. Part of \overline{st} is shown in green, the witness line is shown in blue, and a constraint is shown in orange. (a) cannot occur because the witness line is not a convex chain (invariant 3), and (b) cannot occur because the witness line is crossed by a constraint (invariant 4).

These properties will be proven later, once we have outlined the algorithm. Also note that the witness line is not an actual path in the graph, but rather a guide sent along with the message while routing. [Figure 3.2] gives an example of a witness line. Note that, even though t is blocked

canonical triangle $\Delta_{pp'}$ that is furthest away from p) intersects W_j . Finally, p' is *far* from p if W_j intersects the interior of $\overline{pp'}$ and the sweepline does not cross W_j [Figure 3.3].

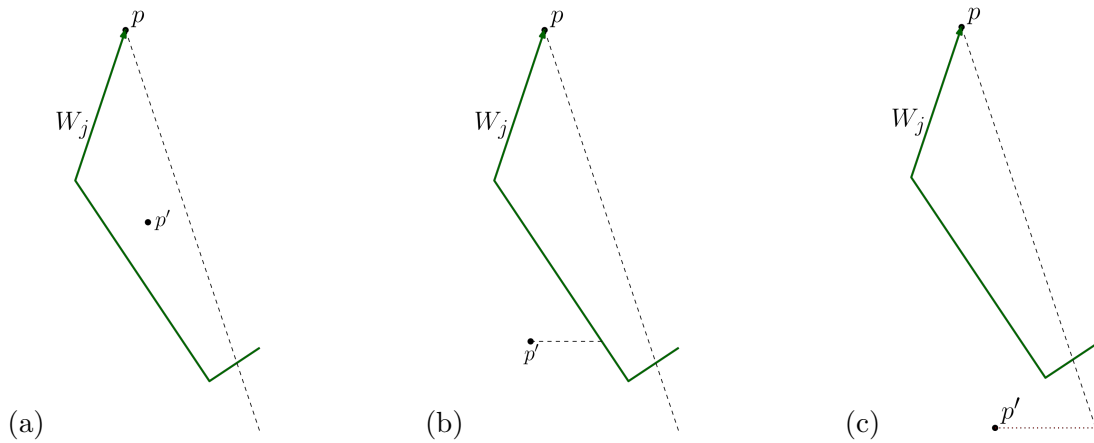


Figure 3.3: Three cases for determining distance from p with respect to W_j . (a) p' is near since its edge from p does not intersect the witness line. (b) p' is intermediate since the sweepline intersects W_j . (c), p' is far since the edge intersects the witness line, and the sweepline does not intersect W_j .

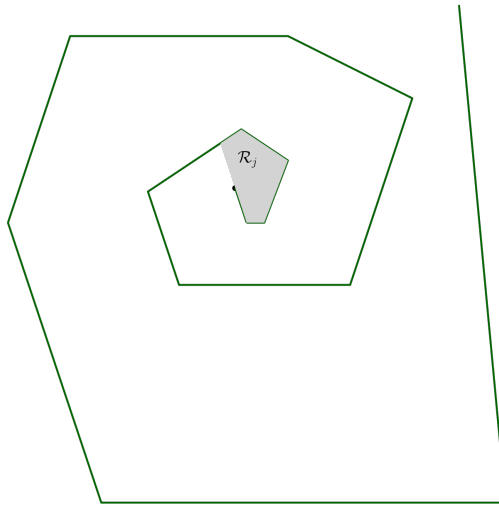


Figure 3.4: An example of the pseudo-interior region \mathcal{R}_j . \mathcal{R}_j is the intersection of the right-side half-planes of each edge of the witness line (where each edge is oriented from t to the current site). These are closed halfplanes, so the current site is also in \mathcal{R}_j . (Note: this witness line is not to scale.)

There is a fifth important property that the witness line maintains in order to keep it from being crossed by constraints. For this one, we would ideally want to define an interior region for the witness line, but of course we cannot do that as it is not a closed figure. We can, however, approximate the idea of an interior by taking advantage of the fact that it is a convex chain. Define

\mathcal{R}_j to be the intersection of all closed halfplanes to the right (recall that we assume we are located to the right of \vec{ts}) of each directed segment e of W_j . In particular, note that $p \in \mathcal{R}_j$. You can think of \mathcal{R}_j as the region bounded by the inner spiral of W_j [Figure 3.4]. With this in mind, we add the fifth property to our list:

1. W_j is simple (i.e., it has no self crossings).
2. W_j is a polygonal chain whose first edge is contained in \overleftarrow{st} . Moreover, all other edges are along some cone boundary.
3. The witness line is a convex chain. That is, three consecutive vertices along W_j form a clockwise turn [Figure 3.1(a)].
4. No constraint can properly cross W_j [Figure 3.1(b)].
5. For all $q \in \mathcal{R}_j$, if an edge $\vec{qq'}$ crosses the witness line but not \vec{ts} , then q' must be far [Figure 3.5].

Note that since q' is far, any near site in the same cone would contradict the existence of the edge $\vec{qq'}$. We prove that all five properties hold for W_0 (i.e. when the witness line is \vec{ts}).

Lemma 3.0.1 *W_0 satisfies all five properties of witness lines.*

Proof Properties 1 and 3 trivially hold as the witness line is a single line segment. Property 2 holds because the first edge is the only edge, and it is \vec{ts} . Property 4 follows from the initial assumption that s and t are visible to each other. Finally, Property 5 is vacuously true, since any edge from q that crosses the witness line would have to cross \vec{ts} . \square

3.1 Algorithm Description

With the above definitions in place, we now describe our routing algorithm. Our routing strategy starts at a site s that is visible to a site t . Without loss of generality, assume that $t \in C_0^s$, and that at any generic step of the algorithm, the current site is on or to the right of the line oriented from t to s (this can be achieved by flipping the coordinate system if needed). Initialize $W_0 = \vec{ts}$.

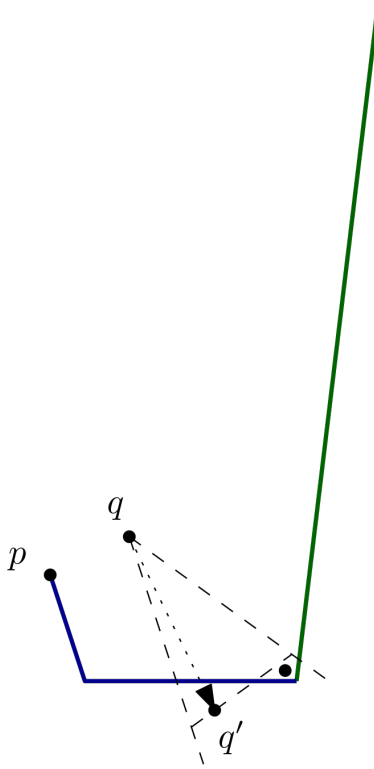


Figure 3.5: This kind of situation is avoided by invariant 5 of the witness line. The edge from q to q' crosses the witness line, so q' must be far. There cannot be any near sites in the cone of q that contains q' , as such a site would contradict the existence of this edge.

In general, after j many steps we are at a site p with witness line W_j . We now define the *scan-cone*. If $p = s$, the scan-cone is simply the cone that contains \vec{ts} . At any other p , by property 2, the last edge of the witness line is along a cone boundary. Following the assumption that p is to the right of \vec{ts} , the *scan-cone* is the cone that has the last edge of the witness line along the clockwise boundary. The scan-cone is the cone whose outgoing edge we check in order to determine how to proceed in the algorithm. Since it is always attached to the witness line, we achieve the goal of “hugging” the witness line by always moving along the scan-cone.

Let $\vec{pp'}$ be the outgoing edge from p in the scan-cone of p (note that p' may not exist). Depending on the distance of p' from p with respect to W_j , we perform one of three steps: a **move step**, a **cross step**, or a **skip step** (these will be formally defined later). We define our algorithm, the Spiral Scan Routing algorithm, as follows:

1. Perform a move step in C_0^s and call the destination vertex p_1 .
2. Suppose we are at a site p_i . If $p_i = t$, end the algorithm. If the scanned cone has an outgoing

edge to a site near to p_i with respect to the current witness line, perform a **move step**. If the outgoing edge crosses st and goes to a site that is both intermediate from p_i and above p_i in y -coordinate (defined by the bisector of C_0), perform a **cross step**. In either case, call the destination p_{i+1} . If neither of these is the case, perform a **skip step**. Update the witness line and repeat this procedure.

In short, at every site p , check the scan-cone, route to a site p' (possibly $p = p'$), and update the witness line. The process will continue until we reach t . Pseudocode summarizing all steps can be found in Algorithm 1.

```

def Spiraling_Scan_Route( $p, t, W$ ):
    if  $p = t$  then
        | Return; ▷ Base case
    Let  $C^*$  be the scan cone w.r.t.  $W$ ;
    if  $\exists$  outgoing edge  $\overrightarrow{pp'}$  in  $C^*$  and  $p'$  is near w.r.t.  $W$  then
        | Update  $W$  by cutting off at the intersection with the sweepline through  $p'$ ,
        |   adding the segment of the sweepline from this intersection to  $p'$ ;
        | Return Spiraling_Scan_Route( $p', t, W$ ); ▷ Move step
    if  $\exists$  outgoing edge  $\overrightarrow{pp'}$  in  $C^*$ ,  $\overrightarrow{pp'}$  crosses  $\overline{st}$ ,  $p'$  is above  $p$ , and  $p'$  is intermediate
    w.r.t.  $W$  then
        | Update  $W$  by cutting  $\overline{st}$  off at its intersection with the sweepline through  $p'$ ,
        |   adding the segment of the sweepline from this intersection to  $p'$ ;
        | Return Spiraling_Scan_Route( $p', t, W$ ); ▷ Cross step
    else
        | Update  $W$  by cutting off at the intersection with the boundary of  $C^*$  and
        |   adding the segment along this boundary;
        | Return Spiraling_Scan_Route( $p, t, W$ ); ▷ Skip step

```

Algorithm 1: The spiraling scan-route algorithm

We now formally define each of these steps, and then show that this algorithm is correct and has a constant spanning ratio.

The first kind of step is the *move step*. If p' is near, follow the edge to p' . Note that, since s sees t , when $j = 0$ (and thus $p = s$), our algorithm will always start with this step. This is the standard way of advancing on the path to t . After moving, the witness line is updated as follows: let x be the intersection of the sweepline through p' with W_j (if there are multiple intersections, x

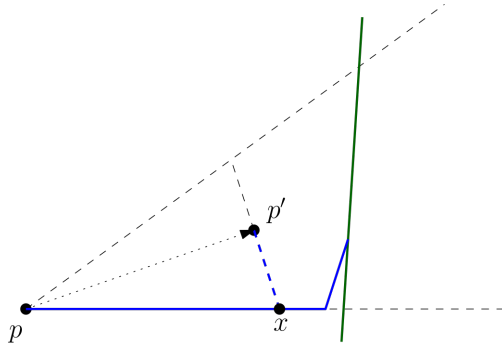


Figure 3.6: During a move step, we move to the next site on the way to t . The witness line is updated by tracing a new edge along the sweepline until it hits the previous sweepline.

is the furthest one from t along W_j). The new witness line W_{j+1} is the portion of W_j from t to x concatenated with the edge $\overline{xp'}$ [Figure 3.6].

We now make sure that all properties are preserved after a move step.

Lemma 3.1.1 *The five properties of the witness line are preserved after a move step.*

Proof Suppose we perform a move step from p with witness line W_j to p' with witness line W_{j+1} . Assume also that all invariants held for all witness lines up to and including W_j . Let $\overline{p'x}$ be the new edge of the witness line.

Property 1 holds since the new edge is traced from p' to its first intersection with W_j , so the witness line stays simple. Property 2 follows from the fact that this is a Θ_{4k+2} graph; the sweeplines are parallel to cone boundaries. Since the new edge is along a sweepline through p' , it is also along one of the cone boundaries of p' .

Since p' is in \mathcal{R}_j , the angle between W_j and $\overline{xp'}$ must be less than π , so Property 3 also holds. A constraint that crosses $\overline{p'x}$ would also have to cross either an edge of W_j or $\overline{pp'}$ [Figure 3.7], none of which are possible (neither W_j by inductive hypothesis, nor $\overline{pp'}$ since it is an edge), so Property 4 holds. Finally, note that \mathcal{R}_{j+1} is a subregion of \mathcal{R}_j . Let there be some site q in \mathcal{R}_{j+1} such that the outgoing edge in C_i^q is to a site q' that is not near. If $\overline{qq'}$ intersects an edge of W_j , there cannot be a near site in C_i^q by the inductive hypothesis. The only way $\overline{qq'}$ can intersect W_{j+1} but not W_j is if it intersects $\overline{p'x}$.

Consider the region bounded by $\overline{pp'}$, $\overline{p'x}$, and the edges removed from W_j after the move step. Call this region H . Note that H is empty because $\overrightarrow{pp'}$ would not have been the outgoing edge from p otherwise. Thus, $\overline{qq'}$ intersecting $\overline{p'x}$ implies that it must also intersect either $\overline{pp'}$ or one

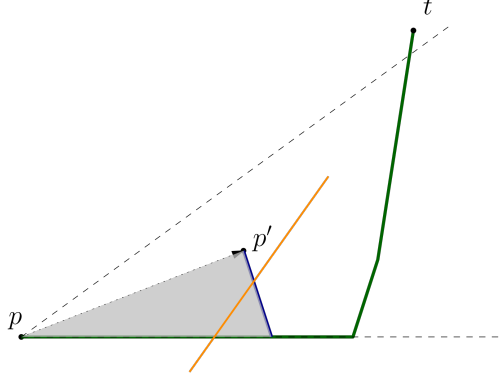


Figure 3.7: Example of the kind of constraint we would have if the new edge of the witness line were crossed by a constraint. Since the gray region is empty, the constraint would have had to cross either the previous witness line or $\overline{pp'}$, none of which are possible.

of the edges of W_j . If it intersects W_j , then again by inductive hypothesis the claim is satisfied. Suppose instead that it intersects $\overline{pp'}$. Consider the boundaries of C_i^q . If only one intersects $\overline{p'x}$, then $p' \in C_i^q$, and the outgoing edge cannot be to a far site. If both intersect $\overline{p'x}$, then call the intersections x_1 and x_2 , where x_1 is closer to p' and x_2 is closer to x . Compare the two canonical triangles Δ_{qx_1} and Δ_{qx_2} , and let the third vertex of the larger one be r [Figure 3.8]. If Δ_{qr} is empty, then there are no sites near q with respect to W_{j+1} in this cone.

Notice that $\angle rx_1x_2 \leq \angle rx_1q \leq \angle pp'x$ and $\angle rx_2x_1 \leq \frac{\pi-\alpha}{2} \leq \angle p'xp$ (recall that α is the angle of the cone at the apex), so $\Delta_{x_1x_2r}$ is contained within H [Figure 3.8]. H is empty, so q' needs to be further than r . Thus, $\Delta_{qq'}$ contains Δ_{qr} , so the latter is empty. All sites near q in C_i^q with respect to W_{j+1} would have to lie in Δ_{qr} , which is empty, so Property 5 holds.

□

The *cross step* is defined as follows: if p' is intermediate, has higher y -coordinate (defined by the bisector of C_0) than p , and lies on the other side of the original line st , then move to p' . The witness line is updated in a similar way as a move step: shoot a ray from p' along the direction of the sweep line until it hits a point x on \overline{st} (recall that this is always possible because p' is intermediate). W_{j+1} is defined as the portion of \overline{st} from t to x and the segment $\overline{xp'}$ [Figure 3.9]. The cross step is essentially a special kind of move step.

We now show that the properties are preserved after a cross step.

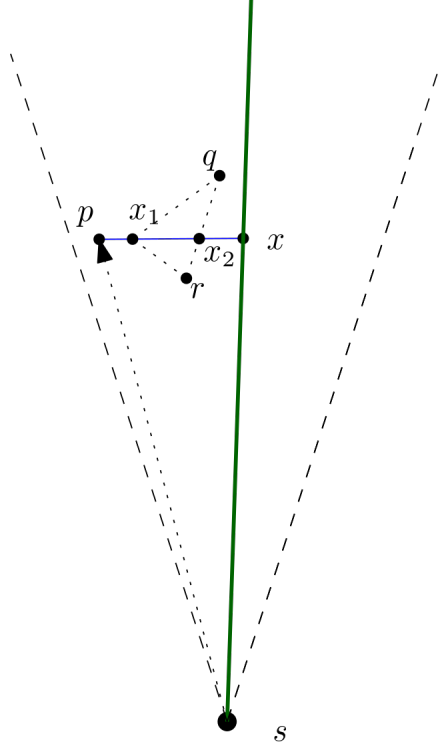


Figure 3.8: Example of the relevant canonical triangles from q after a move step. The green line is part of \overline{st} (so this would be the first step), and \overline{px} is the new edge of the witness line. Note that $\triangle rx_1x_2$ is contained within $\triangle spx$, which is empty. This means that if the endpoint of the outgoing edge from q in the outlined cone is not near, its canonical triangle from q must be larger than that of r .

Lemma 3.1.2 *The five properties of the witness line are preserved after a cross step.*

Proof Suppose we perform a cross step from p with witness line W_j to p' on the other side of st with witness line W_{j+1} . Define $\overline{p'x}$ as the new edge in W_{j+1} . The third invariant follows from the fact that the new witness line consists of exactly two segments. The proofs for all other invariants are identical to the ones for a move step, except that we consider $W_0 = \overline{st}$ instead of W_j [Figure 3.10].

□

The *skip step* occurs when none of the other cases are satisfied. In other words, if the outgoing edge in the scan-cone either does not exist or crosses the witness line, we do not move (except for the specific scenario when we do a cross step). The witness line is updated as follows: the boundary of the scan-cone is formed by two rays, one of which contains the last segment of W_j . Shoot a ray from p along the other boundary until it hits W_j (in Lemma 3.1.3 we will prove that such an intersection always exists). Let x be the first intersection point between the ray and

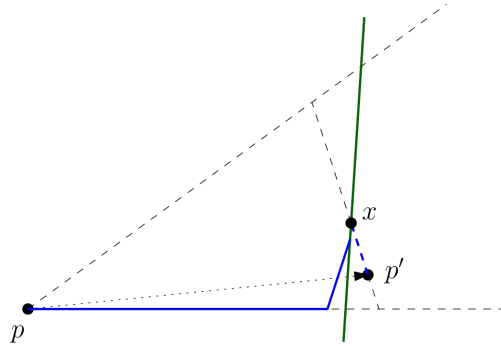


Figure 3.9: During a cross step, cross to the other side of st , updating the witness line in pretty much the same way as a move step, except that the new edge is from p' to \overline{st} , rather than to the previous witness line.

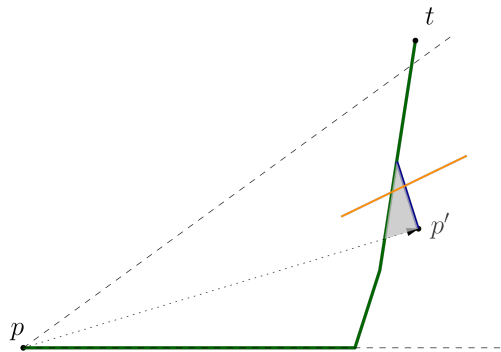


Figure 3.10: Example of the kind of constraint we would have if the new edge of the witness line were crossed by a constraint. Since the gray region is empty, the constraint would have had to cross either \overline{st} or $\overline{pp'}$, none of which are possible.

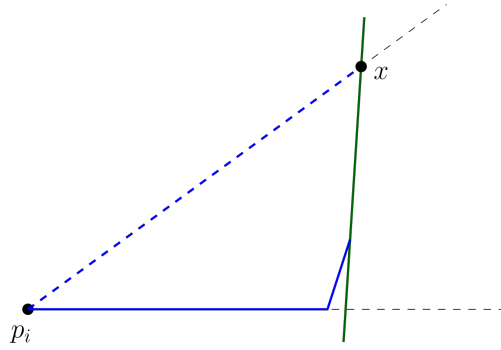


Figure 3.11: During a skip step, we do not move from our current site. The new edge of the witness line is along the boundary of the scan-cone, from p to the previous witness line.

W_j . We create W_{j+1} by replacing the portion of W_j from x to p with the segment \overline{xp} [Figure 3.11]. Although in this case we do not move, we treat it as a move of length zero.

We now show that whenever we perform a skip step, the new witness line is well-defined. Since we have yet to prove that the witness line properties all hold, we will just assume they do for now.

Lemma 3.1.3 *Suppose the witness line properties hold for our current witness line. If we perform a skip step, the ray traced along the boundary intersects the witness line.*

Proof Say we are at p with witness line W_j and we perform a skip step. Suppose the ray never intersects W_j . Since W_j is a convex chain, t must be in the scan cone and it must be near p . If t is visible to p , we reach a contradiction since we would not have performed a move step. If t is not visible, then there must be a constraint blocking it, but then the constraint must have an endpoint in the scan cone that is visible (or if it is blocked by another constraint, we keep finding endpoints until we find one that is not blocked). Again, we reach a contradiction. Thus, the ray must intersect W_j . \square

We will now prove that this last kind of step preserves the invariants.

Lemma 3.1.4 *The five properties of the witness line are preserved after a skip step.*

Proof Suppose we perform a skip step at p and the witness line goes from W_j to W_{j+1} . Say the new edge added during the skip step is \overline{px} . First, we assume the invariants hold for W_j , so we can apply Lemma 3.1.3 to guarantee that x exists.

If there is no site at all in the scan cone, then the polygon created by the scan-cone's intersection with R_j in particular must be empty. Else, the outgoing edge from p in the scan-cone is to a far site p' . Since the only edge of W_{j+1} not in W_j is along the boundary of the scan cone, $\overrightarrow{pp'}$ must intersect an edge in W_j . Since $p \in \mathcal{R}_j$, the existence of p' means that there is no near site in the scan-cone of p , by the inductive hypothesis. So, the intersection of the scan cone with R_j would be empty in this case as well. Thus, during a skip step, the intersection of the skipped cone and \mathcal{R}_j is empty.

Now we show that the invariants still hold after a skip step. Property 1 holds for the same reason as the previous two steps, and Property 2 holds because \overline{px} is defined along the boundary of a cone. Property 3 holds since W_j is a convex chain and p is in \mathcal{R}_j : any segment traced from p to the witness line would maintain convexity. Furthermore, consider the polygon formed by \overline{px} and the edges of W_j removed during the step. Since the interior of this polygon is empty, any constraint that intersects \overline{px} must also intersect another edge of the polygon, but these are all edges of W_j , so this cannot happen [Figure 3.12]. Thus no constraint crosses \overline{px} , and Property 4 holds.

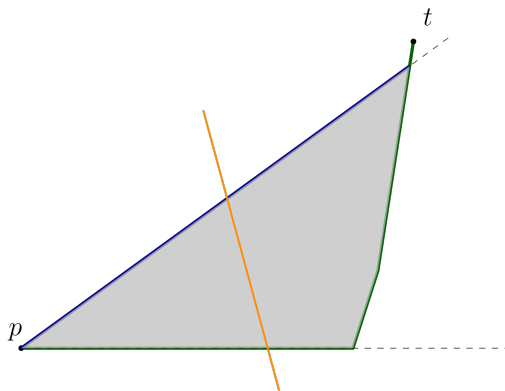


Figure 3.12: Example of the kind of constraint we would have if the new edge of the witness line were crossed by a constraint. Since the gray region is empty, the constraint would have had to cross the previous witness line.

Suppose there is a $q \in \mathcal{R}_{j+1}$ such that C_i^q contains an outgoing edge to a site q' that is not near with respect to W_{j+1} . Note that if $\overline{qq'}$ intersects \overline{px} , then, since the interior of the previously mentioned polygon is empty, $\overline{qq'}$ intersects an edge of W_j . Let $W_{j'}$ be the witness line when this edge is created. \mathcal{R}_j is a subregion of $\mathcal{R}_{j'}$ since $\mathcal{R}_{j'}$ is formed by the intersection of a subset of the halfplanes that form \mathcal{R}_j . Thus $q \in \mathcal{R}_{j'}$, and by the inductive hypothesis, C_i^q has no near sites with respect to W_{j+1} . \square

We wrap up this section by proving that the witness line always has the properties we described.

Theorem 3.1.5 *For any $j \geq 0$, the witness line satisfies the following properties:*

1. W_j is simple (i.e., it has no self crossings)
2. W_j is a polygonal chain whose first edge is contained in \overline{st} . Moreover, all other edges are along some cone boundary.
3. The witness line is a convex chain. That is, three consecutive vertices along W_j form a clockwise turn.
4. No constraint can properly cross W_j .
5. For all $q \in \mathcal{R}_j$, if an edge $\overrightarrow{qq'}$ crosses the witness line but not \overrightarrow{ts} , then q' must be far.

Proof We prove this by strong induction. The base case was proven in Lemma 3.0.1. Suppose the properties hold for all witness lines $W_{j'}$ with $j' \leq j$. Now we either perform a move step, a cross step, or a skip step. After each step, W_{j+1} satisfies the properties by Lemmas 3.1.1, 3.1.2, and 3.1.4, respectively. Thus, the properties hold for all witness lines. \square

3.2 Algorithm Analysis

Since the witness line is sent along with the message during our routing scheme, it is worth finding a bound for the number of edges in the witness line. If during a step we go from p to p' (where p' could be p , if it was a move step), we say that the edge traced in this step was *created by p'* . As an edge case, we say that the first edge of the witness line was created by s . Then the last edge of the current witness line is always created by the current site. With this new way of characterizing edges, we can relate the number of edges in any given witness line to the number of sites in the graph.

Lemma 3.2.1 *Each vertex creates at most one edge of any given witness line.*

Proof Suppose that a witness line has two edges e_1 and e_2 created by the same site p , where e_2 was created later than e_1 . Say W_j is the witness line at the moment e_2 is created. e_1 must be on

W_j . Due to the convexity of the witness line and the fact that t lies on it, this would mean that the witness line crosses either itself or \overline{st} . Neither of these can occur, so there cannot be two such edges. \square

Corollary 3.2.2 *Every witness line has $O(n)$ edges.*

Proof By Lemma 3.2.1, since each vertex creates at most one edge, there cannot be more than n edges. \square

We now work on proving that the algorithm terminates. On top of showing termination, we give a bound on the length of the path taken. We start by proving that any single step from p to p' (where p' could be equal to p) shrinks the witness line by at least $|\overline{pp'}| \cdot \left(1 - 2\sin\left(\frac{\pi}{4k+2}\right)\right)$.

Lemma 3.2.3 *A move step from p to p' shrinks the witness line by at least $|\overline{pp'}| \cdot \left(1 - 2\sin\left(\frac{\pi}{4k+2}\right)\right)$.*

Proof Consider the canonical triangle $\Delta_{pp'}$. Say the sides of this triangle along the cone boundaries have length d , and the sweepline has length h . the length of the edge added is at most h , since the new edge is a part of the sweepline. Also, the part of the witness line removed by this move is at least d . Thus, the witness line shrinks by at least $d - h$. If we let α be the angle at the apex, then $h = 2d\sin(\frac{\alpha}{2})$, so the witness line shrinks by at least $d(1 - 2\sin(\frac{\alpha}{2}))$. Finally, $d \geq |\overline{pp'}|$ and $\alpha = \frac{\pi}{2k+1}$, so we can conclude that the witness line shrinks by at least $|\overline{pp'}| \cdot \left(1 - 2\sin\left(\frac{\pi}{4k+2}\right)\right)$.



Figure 3.13: Two examples illustrating how the witness line is modified after a move step. If the canonical triangle to the new site is illustrated by lengths d and h , we can see that the maximum length that can be added to the witness line is h , when the next site is at the top of the sweepline. Note also that even if the sweepline does not intersect the last edge of the witness line, the length of the witness line lost is no less than d .

\square

Lemma 3.2.4 *A cross step from p to p' shrinks the witness line by at least $|\overline{pp'}| \cdot \left(1 - 2\sin\left(\frac{\pi}{4k+2}\right)\right)$.*

Proof The argument for the cross step is identical to that of a move step. Even though p' is on the other side of the witness line, d and h are the same, so the same bounds apply. \square

Lemma 3.2.5 *The length of the witness line decreases after every skip step.*

Proof Say we are at a site p with witness line W_j , and the boundary of the scan cone that does not contain the last edge of W_j first intersects the witness line at x . We are replacing a portion of the witness line that connects points p and x with the straight segment \overline{px} , so the total length decreases. \square

Corollary 3.2.6 *A skip step from p to p' shrinks the witness line by at least $|\overline{pp'}| \cdot \left(1 - 2\sin\left(\frac{\pi}{4k+2}\right)\right)$.*

Proof Since $p = p'$, $|\overline{pp'}| = 0$. By Lemma 3.2.5, the witness line shrinks during a skip step, so it must shrink by at least a length of 0. Thus, the claim holds. \square

By Lemmas 3.2.3, 3.2.4, and 3.2.5, we can conclude that if we go from p_1 to p_2 , the witness line shrinks by at least $|\overline{p_1p_2}| \cdot \left(1 - 2\sin\left(\frac{\alpha}{2}\right)\right)$. Applying them again, we find that if we go from p_1 to p_2 to p_3 , the witness line shrinks by at least $|\overline{p_1p_2}| \cdot \left(1 - 2\sin\left(\frac{\alpha}{2}\right)\right) + |\overline{p_2p_3}| \cdot \left(1 - 2\sin\left(\frac{\alpha}{2}\right)\right)$, or $(|\overline{p_1p_2}| + |\overline{p_2p_3}|) \cdot \left(1 - 2\sin\left(\frac{\alpha}{2}\right)\right)$.

In general, if we follow a sequence of sites $\{p_1, p_2, \dots, p_m\}$, the witness line shrinks by at least $\left(\sum_{i=1}^{m-1} |\overline{p_i p_{i+1}}|\right) \left(1 - 2\sin\left(\frac{\alpha}{2}\right)\right)$. Note that the term on the left of this expression is just the length of the path taken from p_1 to p_m , so if we were to travel a path of length $\frac{|\overline{st}|}{\left(1 - 2\sin\left(\frac{\alpha}{2}\right)\right)}$, the witness line would shrink by a length of $|\overline{st}|$. The initial witness line has length $|\overline{st}|$, so after traversing a path of this length, the witness line would have length 0. Since the witness line goes from t to our current location, this can only mean we reach t . We summarize these observations in the following theorem.

Theorem 3.2.7 *For any set S of n sites and set T of linear constraints with endpoints in S , there exists a 1-local and competitive routing strategy in the constrained Θ_{4k+2} -graph of S and T . This strategy can route between any two visible sites using $O(n)$ -memory by traversing a length of at*

$$\text{most } \frac{|\overline{st}|}{1 - 2\sin\left(\frac{\pi}{4k+2}\right)}.$$

Proof By definition, the Spiral Scan algorithm is 1-local and runs on Θ_{4k+2} graphs. The information sent between nodes consists of just the witness line, which by Corollary 3.2.2 consists of an $O(n)$ number of edges, so the algorithm uses $O(n)$ memory. Finally, since every move step or cross step length d shortens the witness line by at least $d \cdot \left(1 - 2 \sin\left(\frac{\pi}{4k+2}\right)\right)$, the algorithm needs to travel no more than $\frac{|\overline{st}|}{\left(1 - 2 \sin\left(\frac{\pi}{4k+2}\right)\right)}$ in order for the witness line to shrink by $|\overline{st}|$. In other words, the path taken is within a factor of $\frac{1}{1 - 2 \sin\left(\frac{\pi}{4k+2}\right)}$ of the distance $|\overline{st}|$. \square

Below we conclude this section with a brief table of values of $\frac{1}{1 - 2 \sin\left(\frac{\pi}{4k+2}\right)}$ for various values of k for reference.

Number of cones	α	$\frac{1}{\left(1 - 2 \sin\left(\frac{\alpha}{2}\right)\right)}$
10	$\frac{\pi}{5}$	2.618
14	$\frac{\pi}{7}$	1.802
18	$\frac{\pi}{9}$	1.532

Chapter 4

Future Work

The spiral scan algorithm used $O(n)$ memory because each site must know the entire witness line before making a step. The witness line could have $n - 1$ edges in the worst case (every site creates an edge except for t) [Figure 4.1]. We ask ourselves if we really need to know all the edges and not just the first few. This is a natural question, since we only need one intersection with the witness line to determine whether a site is near or far, as shown in Figure 4.2. In order to reduce the memory needed, we think back to the primary purpose of the witness line. The witness line is meant to guide us around constraints that may be blocking us from t . If the witness line starts to spiral, we may be getting a guide that is unnecessarily convoluted.

If we are at a site p , and t is in C_i^p , then there are only so many cones where the endpoints of the constraint can be. We can check all of these cones by sweeping from C_{-i}^p to C_i^p (where C_{-i}^p is opposite C_i^p). This led us to consider cutting off the witness line when the scan cone goes beyond C_{-i}^p . However, this approach required us to allow constraints to cross the witness line in certain scenarios, which sacrificed the competitiveness of the algorithm.

Another strategy we are considering is to use both incoming and outgoing edges. As noted earlier, both the Spiral Scan algorithm and the variant mentioned above use only outgoing edges. These edges provide information on empty regions that we use to define the algorithm. Nonetheless, Θ -graphs have more information that we have not yet used. Namely, for each site p we can consider the incoming edges and the empty regions that come from them. We seek to use both incoming and outgoing edges to develop a competitive algorithm that uses a constant-sized message. We are currently exploring ways to implement this idea.

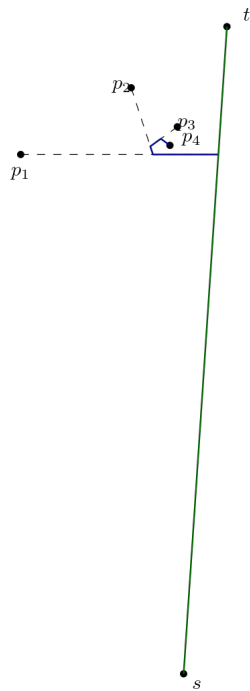


Figure 4.1: A small example of how we can adversarially place points so the witness line has $O(n)$ edges. At p_4 , the witness line has 5 edges. We can continue placing new sites in the scanned cone of the last site to have n sites and $n - 1$ edges in the witness line.

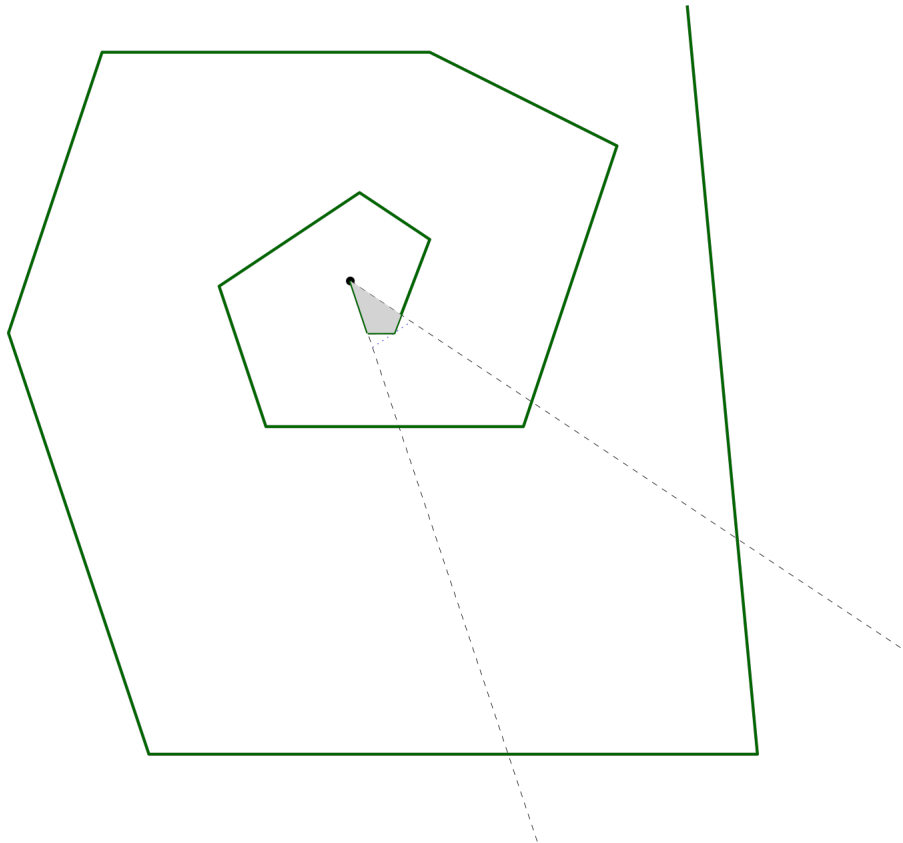


Figure 4.2: Example of a witness line that spirals a lot (scale adjusted slightly for visibility). Note that the information used by the algorithm at the current site is constrained to what is in the region. We will need to spiral counterclockwise in order to “unwind” the witness line. We hope to avoid this by not spiraling in the first place.

Bibliography

- [BDCHS19] Prosenjit Bose, Jean-Lou De Carufel, Darryl Hill, and Michiel Smid. On the spanning and routing ratio of theta-four. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2361–2370. SIAM, 2019.
- [BFVRV12] Prosenjit Bose, Rolf Fagerberg, André Van Renssen, and Sander Verdonschot. Competitive routing in the half- θ 6-graph. In *Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms*, pages 1319–1328. SIAM, 2012.
- [BFVRV15] Prosenjit Bose, Rolf Fagerberg, André Van Renssen, and Sander Verdonschot. Competitive local routing with constraints. In *International Symposium on Algorithms and Computation*, pages 23–34. Springer, 2015.
- [BVR14] Prosenjit Bose and André Van Renssen. Upper bounds on the spanning ratio of constrained theta-graphs. In *Latin American Symposium on Theoretical Informatics*, pages 108–119. Springer, 2014.
- [Cla87] Ken Clarkson. Approximation algorithms for shortest path motion planning. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 56–65, 1987.
- [KG92] J Mark Keil and Carl A Gutwin. Classes of graphs which approximate the complete euclidean graph. *Discrete & Computational Geometry*, 7(1):13–28, 1992.
- [RS91] Jim Ruppert and Raimund Seidel. Approximating the d-dimensional complete euclidean graph. In *Proceedings of the 3rd Canadian Conference on Computational Geometry (CCCG 1991)*, pages 207–210, 1991.

- [STMZ18] Weisheng Si, Quincy Tse, Guoqiang Mao, and Albert Y Zomaya. On the performance of greedy forwarding on yao and theta graphs. *Journal of Parallel and Distributed Computing*, 117:87–97, 2018.
- [Xia13] Ge Xia. The stretch factor of the delaunay triangulation is less than 1.998. *SIAM Journal on Computing*, 42(4):1620–1659, 2013.